Mikrokontrolery STR91x od podstaw, część 1

„Sercem" mikrokontrolerów STR91x jest jednostka centralna ARM966E–S, której historia sięga roku 1997. Układy STR91x to pierwsza i na razie jedyna rodzina mikrokontrolerów wyposażonych w taki rdzeń, co powoduje że w niektórych aplikacjach są one niezastąpione.

W artykule przedstawiamy skrótowo architekturę rdzenia ARM966E–S, najważniejsze zagadnienia związane z listą rozkazów, a także podstawowe informacje o aplikacji.

W kolejnych odcinkach skupimy się na omówieniu narzędzi programowych i przykładowych projektach.

Postanowiliśmy przygotować ekspresowy kurs poświęcony mikrokontrolerom STR91x gównie z powodu wielu zapytań o to czym różnią się one od popularnych wśród konstruktorów mikrokontrolerów wyposażonych w rdzeń ARM7TDMI. Ponieważ lubimy odpowiadać przykładami, to po krótkim wstępie „teoretycznym" zajmiemy się prezentacją przykładów, z których część zilustruje najważniejsze różnice.

W mikrokontrolerach STR91x rdzeń ARM966E–S ma „na sztywno" ustalony porządek bajtów *little–endian*, o czym musimy pamiętać podczas ustawiania opcji kompilatora. Bardzo ciekawą cechą ARM–ów jest możliwość wykonywania każdej instrukcji w sposób warunkowy. Najstarsze cztery bity kodu rozkazu są porównywane z bitami flag warunków w rejestrze CPSR. Jeżeli bity te nie są zgodne, dana instrukcja nie jest wykonywana (w jej miejsce do potoku wstawiany jest rozkaz NOP). Wykonywanie każdej instrukcji w sposób warunkowy pozwala w dużej liczbie przypadków na uniknięcie używania instrukcji rozgałęzień, których wykonanie powoduje ponowne zapełnianie potoku nowymi instrukcjami i dodatkową stratą czasu. Do większości mnemoników asemblera w trybie ARM można dodać sufiks z mnemonikami warunkowymi. Listę mnemoników przedstawiono w **tab. 1**.

Przykładowo instrukcja ADDMI R1,R2,#0x8000000 dodaje do rejestru R1 zawartość R2+0x80000000 tylko wtedy, gdy wynikiem wykonania poprzedniej operacji było ustawienie bitu N w rejestrze CPSR, w przeciwnym razie zawartość rejestru R1 pozostanie niezmieniona. Procesor ARM ma architekturę *Load/Store*, więc wszystkie operacje muszą być wykonywane na rejestrach. Aby wykonać jakąś operację na danej znajdującej się w pamięci musimy jej zawartość przesłać do rejestru, wykonać żądaną operację, a następnie wynik operacji przesłać ponownie do pamięci. Lista in-

Potok w ARM9

Zastosowany w rdzeniu ARM966E–S potok działa najefektywniej, gdy program wykonywany jest cyklicznie i nie występują rozgałęzienia zmieniające kolejność wykonania programu. W przypadku wystąpienia rozgałęzienia, potok musi zostać oczyszczony z niepotrzebnych instrukcji występujących za rozgałęzieniem i wypełniony nowymi instrukcjami, co zajmuje dodatkowy czas powodując wydłużenie wykonania rozkazu. Tak więc im dłuższy potok posiada mikrokontroler, tym więcej czasu zostanie stracone w przypadku wystąpienia rozgałęzienia. W odróżnieniu od innych powszechnie znanych mikroprocesorów, w ARM–ach prawie każda instrukcja może być wykonywana warunkowo, w zależności od stanu znaczników zmienionych przez poprzednią instrukcję. Znakomicie ułatwia to tworzenie kodu bez rozgałęzień, zapobiegając stracie czasu na ponowne zapełnienie potoku. W innych popularnych architekturach tylko instrukcje warunkowe mogą sprawdzać stan znaczników, w efekcie czego program nie jest wykonywany sekwencyjnie, co w przypadku przetwarzania potokowego jest rozwiązaniem wysoce nieoptymalnym. W mikrokontrolerach STR91x wprowadzono również dodatkowy mechanizm kolejki PFQ *(Prefetch Queue)* oraz pamięci rozgałęzień BC (*Branch Cache*) pozwalający dodatkowo zminimalizować problem straty czasu na ponowne zapełnianie potoku w przypadku wystąpienia rozgałęzienia. Będzie o tym mowa w kolejnym rozdziale poświęconym mikrokontrolerowi STR91x. Licznik rozkazów PC (R15) wskazuje na osiem bajtów do przodu w stosunku do bieżącej instrukcji, czyli na instrukcje aktualnie pobieraną. Należy o tym pamiętać przy adresowaniu względnym odnoszącym się do licznika rozkazów.

Rdzeń ARM966E–S obsługuje wspólną przestrzeń adresową dla rozkazów i danych, w przeciwieństwie do mikrokontrolerów 8051 czy AVR, które posiadają rozdzieloną przestrzeń adresową. Dostęp do pamięci możliwy jest poprzez jedyne instrukcje operujące na pamięci: ładowania danych z pamięci do rejestru (LD), oraz zapisu z rejestru do pamięci (ST). Instrukcje te operują na 8, 16 lub 32 bitach danych, przy czym procesor może być skonfigurowany tak aby zapisywał dane w porządku *big–endian* lub *little–endian*.

Tab. 1. Pole warunków instrukcji ARM

Mnemonik	Flagi warunków	Działanie
EQ	Z=1	=
NE	Z=0	!=
CS	C=1	>= (liczba bez znaku)
CC	C=0	< (liczba bez znaku)
MI	N=1	Ujemny
PL	N=0	Dodatni lub 0
VS	V=1	Przepełnienie
VC	V=0	Brak przepełnienia
HI	C=1 i Z=0	> (liczba bez znaku)
LS	C=0 i Z=1	<= (liczba bez znaku)
GE	N=V	>=
LT	N!=V	<
GT	Z=0 i (N=Y)	>
LE	Z=1 lub (N!=V)	<=
AL.	Bez znaczenia	Zawsze (można pominąć)

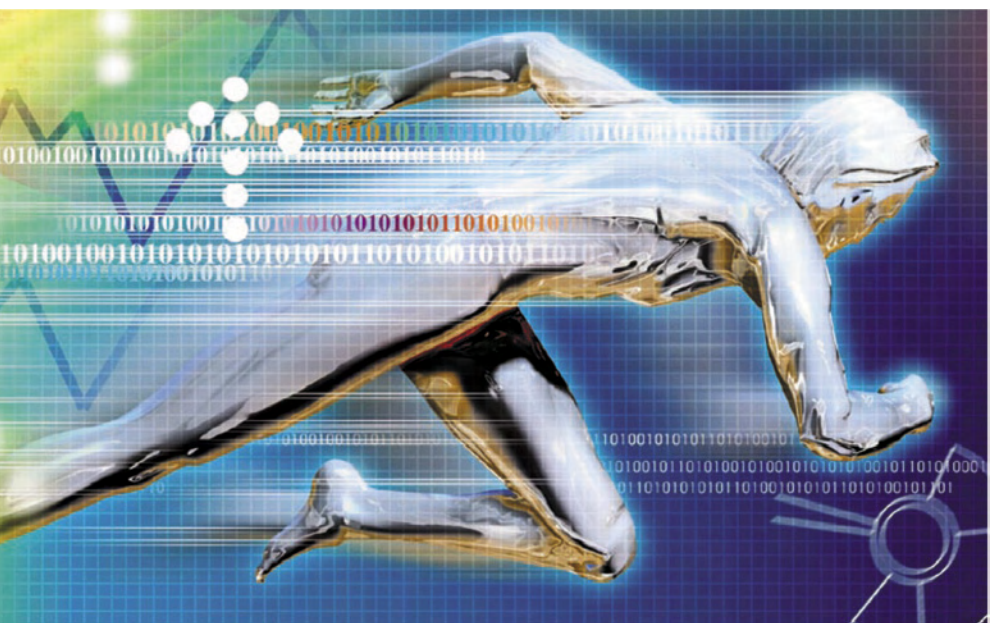
Rdzeń ARM966E–S
Układy STR91x należą do elitarnego grona szybkich mikrokontrolerów, charakteryzujących się jednocześnie niewielkim poborem mocy. Zastosowany w nich rdzeń ARM966E–S powstał w wyniku „wyrzucenia" z podstawowego rdzenia ARM9 jednostki zarządzania pamięcią MMU oraz zastąpienie pamięci cache pamięcią TCM.

strukcji nie daje możliwości wykonywania instrukcji arytmetycznych i logicznych bezpośrednio na komórkach pamięci jak ma to miejsce w mikroprocesorach CISC. Załadowanie zawartości komórki pamięci do rejestru umożliwia instrukcja *LDR{warunek}[B/BT/SB/H/SH] Rd,<adr_mode>*. Instrukcja może być wykonana warunkowo, co określa opcjonalne pole {*warunek*}, które może być wypełnione zgodnie z tab. 1, wówczas instrukcja będzie wykonana w zależności od wyniku poprzedniej operacji arytmetycznej. Analogiczna instrukcją umożliwiającą zapisanie zawartości pamięci do rejestru jest instrukcja *STR{warunek}[B/BT/SB/H/SH] Rd,<adr_mode>*. Parametry [B/BT/SB/H/HS] , określają długość słowa (B – bajt, H – połowa słowa) o interpretację danych pobieranych z pamięci (S – liczba ze znakiem). Adres komórki pamięci jest adresowany za pomocą pola *<adr_mode>*, gdzie pamięć może być adresowana w sposób:

- pośredni rejestrowy [Rn],
- pośredni rejestrowy preindeksowany z przesunięciem natychmiastowym [Rn,± #offset],
- pośredni rejestrowy preindeksowany z przesunięciem zawartym w rejestrze [Rn,± Rm],
- pośredni rejestrowy preindeksowany z przesunięciem w rejestrze i skalowaniem [Rn,±Rm, OP], gdzie OP jest operacją logiczną LSL/LSR/ASR/ROR/RRX na rejestrze RM,
- pośredni rejestrowy postindeksowany z przesunięciem natychmiastowym [Rn],± #offset,
- pośredni rejestrowy postindeksowany z przesunięciem zawartym w rejestrze [Rn,± Rm],
- pośredni rejestrowy preindeksowany z przesunięciem w rejestrze i skalowaniem [Rn],±Rm, OP, gdzie OP jest operacją logiczną LSL/LSR/ASR/ROR/RRX na rejestrze Rm.

Oprócz instrukcji umożliwiających odczyt lub zapis pojedynczej komórki pamięci istnieją także rozkazy umożliwiające odczytanie lub zapisanie bloku rejestrów do pamięci. Po zapisaniu lub odczytaniu bloku rejestrów automatycznie zwiększają lub zmniejszają one rejestr indeksujący pamięć. Do przesłania bloku pamięci do rejestrów służy instrukcja *LDM{warunek}<tryb> Rn,<reglist>* , nastomiast przesłanie bloku rejestrów do pamięci służy instrukcja *STM{warunek}<tryb> Rn,<reglist>*.

Pole *<reglist>* określa listę rejestrów jakie mają zostać pobrane/wysłane do pamięci, natomiast pole *<tryb>* określa sposób adresowania za pomocą rejestru bazowego,



Rys. 1. Format instrukcji przetwarzania danych

gdzie: IA powoduje zwiększenie rejestru bazowego Rn, a następnie przesłanie danych, IB powoduje zwiększenie rejestru bazowego Rn po wykonaniu transferu danych, DA powoduje zmniejszenie rejestru bazowego po wykonaniu transferu danych, DB powoduje zmniejszenie rejestru bazowego przed wykonaniem transferu danych. Instrukcje transferu wielu rejestrów do pamięci są bardzo często wykorzystywane do tworzenia struktury stosu w pamięci, wówczas do adresowania najczęściej wykorzystuje się rejestr R13 z uwagi na to, że posiada on swoją oddzielną kopię dla każdego trybu ochrony. Wykorzystując wyżej wspomniane instrukcję możemy pracować z dowolnym typem stosu, a jego organizacja nie jest ograniczona architekturą mikroprocesora, tylko implementacją kompilatora lub systemu operacyjnego. Instrukcje rozgałęzień również mają oddzielną architekturę niż w innych mikrokontrolerach i potrafią wykonać skok w przód lub w tył o 32 MB od bieżącej instrukcji. Podstawową instrukcją skoku jest *B{warunek} <adres>*, której wykonanie powoduje skok pod wskazany adres. Modyfikacją instrukcji B jest instrukcja *BL{warunek} <adres>*, która działa tak samo jak instrukcja B, ale dodatkowo zapisuje zawartość PC+4 w rejestrze LR, co umożliwia realizację mechanizmu podprogramów, bez konieczności używania stosu. Używając wraz z instrukcją B i BL mnemoników warunkowych można wykonywać skoki warunkowe oraz warunkowe wywołania podprogramów. Specjalną odmianą instrukcji B i BL są instrukcje BX i BLX, które oprócz skoku dodatkowo zapewniają przejście pomiędzy trybami Thumb i ARM. Instrukcje te są jedynym dozwolonym sposobem na zmianę trybu pracy.

Kolejnym typem instrukcji są instrukcje przetwarzania danych, których format w ogólnej postaci przedstawiono na **rys. 1**.

Każda instrukcja przetwarzania danych posiada rejestr przeznaczenia (Rd) oraz dwa rejestry zawie-

rające argumenty. Pierwszy operand musi być rejestrem (Rn) natomiast drugi operand *<operand2>* może być rejestrem lub wartością stałą. Dodatkowo jako czwarty argument możemy podać przesunięcie logiczne do 32 bitów, które nie powoduje wydłużenia czasu wykonania rozkazu. Flaga „S" informuje czy rozkaz ma zaktualizować flagi warunkowe w CPSR czy też nie. Jeżeli jako rejestr przeznaczenia wyspecyfikowany jest rejestr PC (R15) flaga „S" ma dodatkowe znaczenie polegające na skopiowaniu rejestru SPSR z bieżącego trybu ochrony do rejestru CPSR. Instrukcje te używane są do powrotu z podprogramu obsługi wyjątku. Do najczęściej wykorzystywanych instrukcji przetwarzających dane należą:

AND{warunek}{S} Rd,Rn,<operand2>– logiczny AND

EOR{warunek}{S},Rd,Rn,<operand2> – exclusive–OR

SUB{warunek}{S},Rd,Rn,<operand2> – odejmowanie

RSB{warunek}{S},Rd,Rn,<operand2> – odejmowanie w odwróconej kolejności argumentów.

ADD{warunek}{S},Rd,Rn,<operand2> – dodawanie

ADC{warunek}{S},Rd,Rn,<operand2> – dodawanie z przeniesieniem

SBC{warunek}{S},Rd,Rn,<operand2> – odejmowanie z pożyczką

RSB{warunek}{S},Rd,Rn,<operand2> – odejmowanie z pożyczką w odwróconej kolejności argumentów

TST{warunek} Rn,<operand2> – testowanie AND

TEQ{warunek} Rn,<operand2> – testowanie XOR

CMP{warunek} Rn,<operand2> – porównanie

CMN{warunek} Rn,<operand2> – odwrotne porównanie

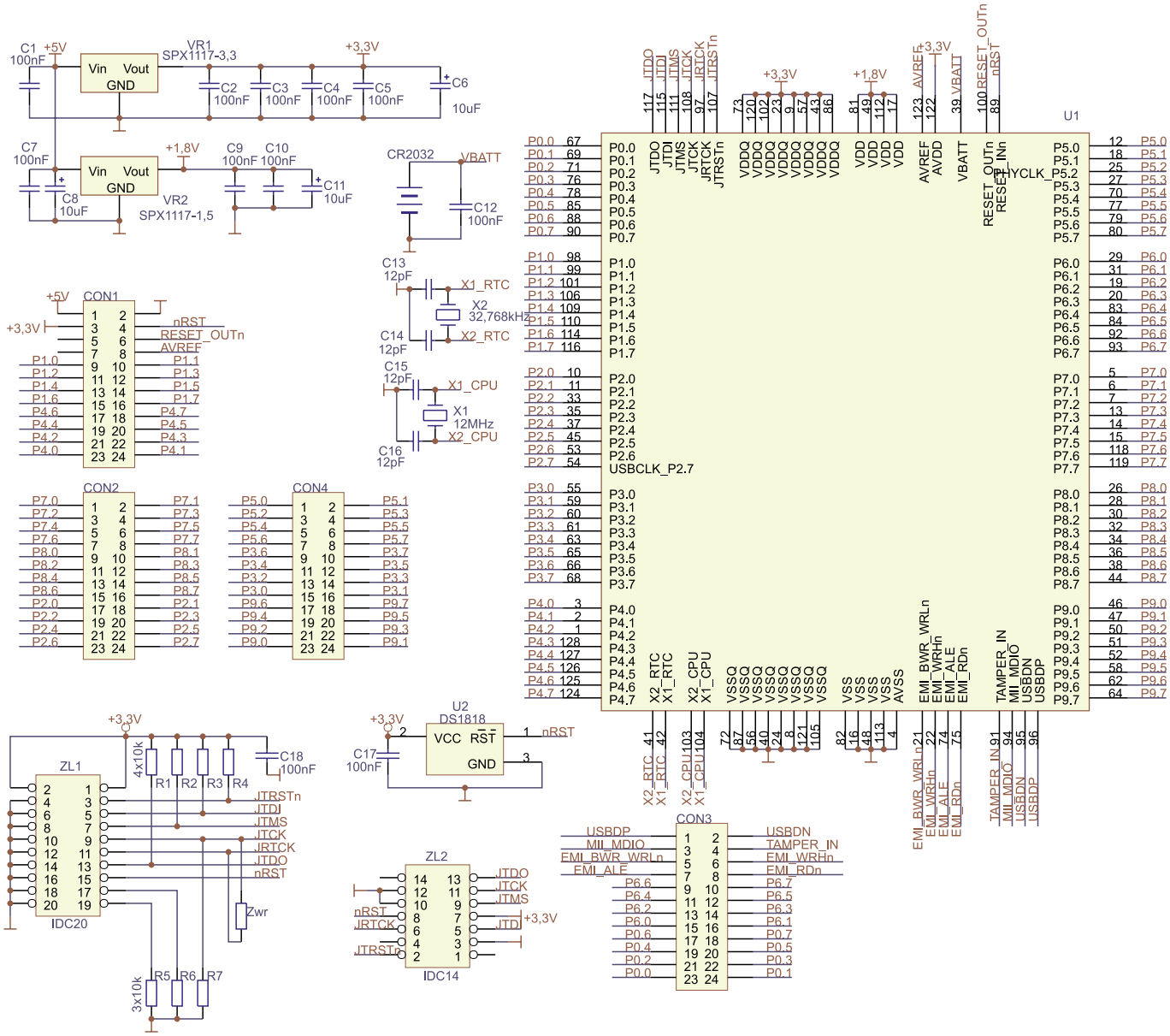
ORR{warunek} Rn,<operand2> – logiczny OR

MOV{warunek}{S} Rd,<operand2> – przeniesienie argumentów

BIC{warunek} Rn,<operand2> – kasowanie wybranego bitu

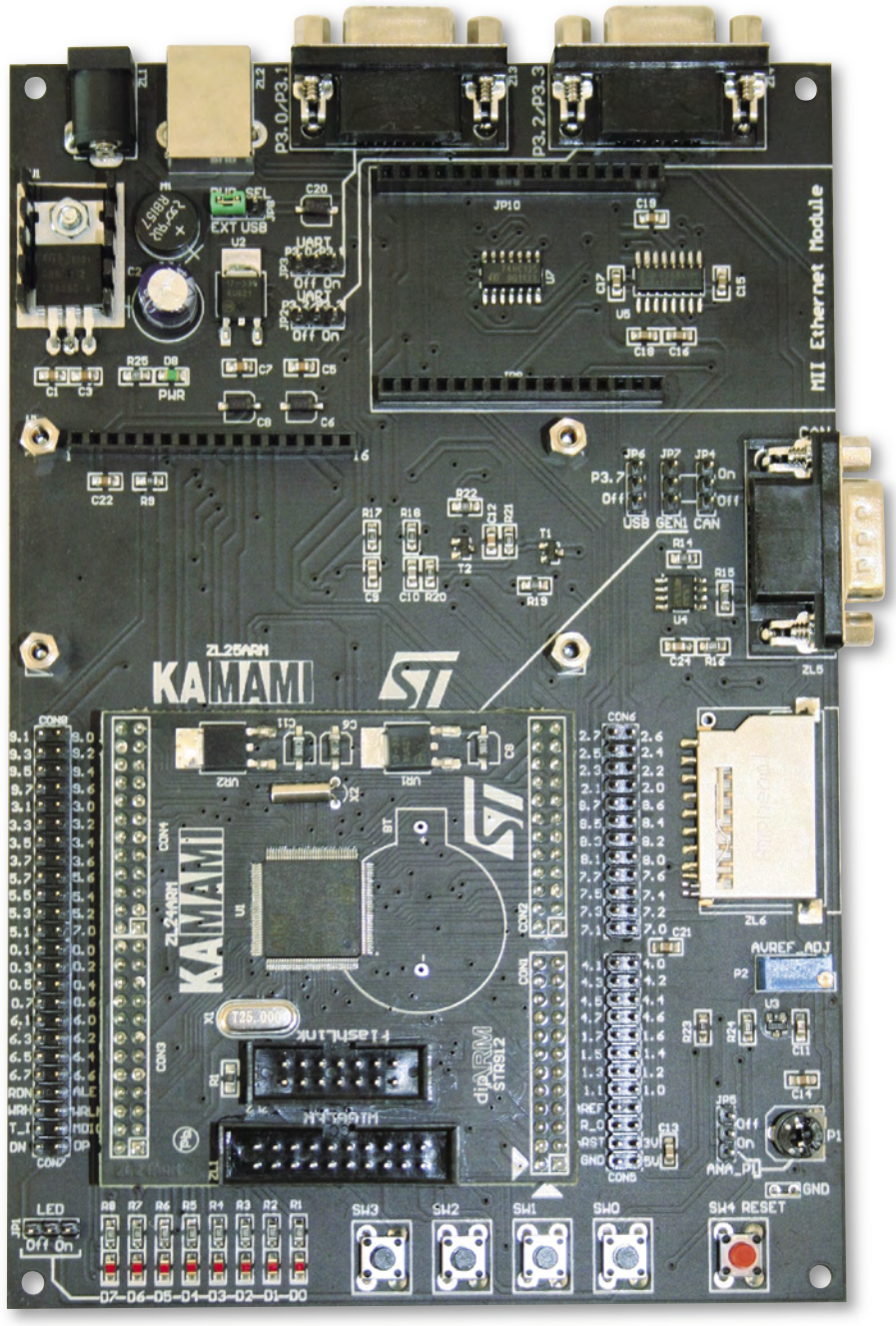
MVN{warunek}{S} Rd,<operand2> – przenieś zanegowany (Rd:=0xFFFFFFFF XOR Operand2)

Ponieważ wszystkie rejestry są ortogonalne (równouprawnione), możemy użyć dowolnych instrukcji operujących bezpośrednio na liczniku rozkazów PC (R15), co jest użyteczne w przypadku, gdy adres



Rys. 2. Podstawowa aplikacja mikrokontrolera z serii STR912

skoku znajduje się poza zasięgiem instrukcji B (*Branch*). Dostęp do rejestrów stanu procesora CPSR,SPSR jest możliwy za pomocą dwóch instrukcji: *MSR{warunek}<PSR>_<pole>,Rm* oraz *MRS Rd,<PSR>*, które umożliwiają przesłanie zawartości CPSR i SPSR do rejestrów ogólnego przeznaczenia. Zmiana trybu ochrony procesora lub załączenie i wyłączenie przerwań możliwe jest tylko z uprzywilejowanego trybu ochrony poprzez modyfikacje rejestru CPSR. Bardzo interesującą instrukcją jest instrukcja przerwania programowego *SWI{warunek}<wartość>*. Jej działanie polega na wygenerowaniu wyjątku przerwania programowego SWI w momencie jej wystąpienia. Jako argument SWI można podać 24–bitową liczbę, którą w podprogramie obsługi wyjątku można odczytać i na jej podstawie określić czynności jakie ma wykonać wyjątek. Instrukcja SWI powoduje zmianę trybu ochrony na tryb *Supervisor* i często za jej pomocą wykonuje się odwołania do funkcji systemu operacyjnego. Pracując bez systemu operacyjnego będziemy mogli ją wykorzystać do przejścia z trybu użytkownika w tryb *Supervisor* w celu np. zablokowania systemu przerwań. Oprócz wyżej przedstawionych instrukcji procesor posiada także instrukcje mnożenia liczb ze znakiem lub bez znaku, oraz instrukcję charakterystyczną dla procesorów DSP – mnożenie z akumulacją. Cechą charakterystyczną ARM–ów są trójargumentowe rozkazy np. *ADDS R0,R1,R2*. Gdzie do rejestru R0 wpisywana jest zawartość R1+R2 w innych mikroprocesorach rozkazy najczęściej operują na dwóch argumentach gdzie operacje wykonuje się na rejestrze przeznaczenia np. dla mikroprocesora x86 instrukcja *ADD EAX,ECX* do rejestru EAX dodaje zawartość EAX + ECX. Unikalną cechą listy instrukcji ARM–ów jest także łączenie operacji przesunięcia i obrotu z instrukcjami arytmetycznymi, logicznymi lub rozkazami przesłań. Jako czwarty argument tych rozkazów możemy wyspecyfikować o ile bitów należy obrócić lub przesunąć zawartość rejestru – np. *ORR r1,r2,r3,lsl #4* powoduje zapis do rejestru R1 sumy logicznej rejestru R2 z rejestrem R3 przesuniętym o 4 bity w lewo. Rdzeń ARM966E–S posiada także dodatkowe instrukcję



Fot. 3. Zestaw ZL24ARM+ZL25ARM, będący platformą sprzętową kursu

przyśpieszające obliczenia w algorytmach DSP. Podobnie jak pozostałe instrukcje mogę one operować na dowolnych rejestrach procesora i można je wykonywać warunkowo. Do grupy tych instrukcji należą m.in.:

QADD {*warunek*} *Rd,Rn,<operand2>* – dodawanie z nasyceniem Rd=SAT(Rm+Rn)

QDADD {*warunek*} *Rd,Rn,<operand2>* – dodawanie z podwójnym nasyceniem Rd=SAT(Rm+SAT(Rn*2))

QSUB {*warunek*} *Rd,Rn,<operand2>* – odejmowanie z nasyceniem Rd=SAT(Rm+Rn)

QDSUB {*warunek*} *Rd,Rn,<operand2>* – odejmowanie z podwójnym nasyceniem Rd=SAT(Rm+SAT(Rn*2))

SMULxy {*warunek*} *Rd,Rm,RS* – mnożenie i akumulacja ze znakiem 16*16 bitów

Opisane instrukcje są dostępne w 32–bitowym trybie ARM. Umiejętne stosowanie instrukcji z wykorzystaniem flag warunkowych umożliwia uzyskanie dużej wydajności dzięki pominięciu występowania rozgałęzień, a co za tym idzie brakiem konieczności ponownego wypełniania potoku w wyniku wystą-

pienia rozgałęzienia. W niektórych aplikacjach nie zawsze jest potrzebna duża moc obliczeniowa, natomiast liczy się niewielki rozmiar kodu programu. Właśnie dla takich aplikacji został zaprojektowany 16–bitowy tryb Thumb, w którym lista instrukcji jest zdecydowanie bardziej podobna do listy klasycznego mikroprocesora. W trybie Thumb nie ma możliwości warunkowego wykonania każdej z instrukcji. Podobnie jak w klasycznych mikrokontrolerach skok warunkowy w określone miejsce umożliwia tylko instrukcja rozgałęzienia *B{warunek} <adres>*. W trybie Thumb wprowadzono także klasyczne wersje instrukcji *PUSH <rejestry_dolne>* oraz *POP <rejestry_dolne>*. Nie ma również możliwości wprowadzenia do rozkazu dodatkowego argumentu umożliwiającego wykonywanie przesunięć logicznych. Instrukcje przesunięć logicznych mogą być wykonywane tylko w postaci osobnych operacji LSLR/LSRS/ASRS/RORS. Natomiast pozostałe instrukcje arytmetyczne i logiczne są w zasadzie identyczne jak w 32–bitowym trybie ARM.

Sprzęt

Stosowanie mikrokontrolerów STR91x we własnych urządzeniach nie jest bardziej skomplikowane niż prostych mikrokontrolerów 8–bitowych, jak na przykład AVR czy 8051. Wszystkie 32–bitowe magistrale mikrokontrolera zawarte są w jego wnętrzu, a sam mikrokontroler aby prawidłowo mógł działać potrzebuje dosłownie kilku elementów. Schemat najprostszej aplikacji mikrokontrolera STR912 (jest to schemat elektryczny zestawu ZL24ARM firmy *kamami.pl*) przedstawiono na **rys. 2**.

Mikrokontrolery STR91x wymagają dwóch napięć zasilających o wartości 1,8 oraz 3,3 V, które są dostarczane przez stabilizatory SPX1117–1.8 (VR2), oraz SPX1117–3.3 (VR1). Napięcia te są dołączone odpowiednio do linii VDDQ (3,3), oraz VDD (1,8) zasilając część cyfrową mikrokontrolera. Część analogowa posiada oddzielne linie zasilające AVDD, które dołączone są bezpośrednio do napięcia 3,3 V części cyfrowej. W przypadku, wykonywania dokładniejszych pomiarów należy część analogową oddzielić od części cyfrowej chociażby za pomocą prostego filtru LC. Linia V_{ref} dostarczająca napięcie referencyjne dla wbudowanego przetwornika A/C, jest wyprowadzona na zewnętrzne złącze, do którego możemy dołączyć precyzyjne źródło referencyjne wykonane np. na układzie TL431. Pomimo że mikrokontroler jest zasilany napięciem 3,3 V do linii portów GPIO można podłączać napięcie o wartości 5 V, co daje możliwość wykorzystywania układów zasilanych tym napięciem bez dodatkowych układów pośredniczących. Do taktowania mikrokontrolera wykorzystano wewnętrzny generator kwarcowy, do którego wejść (XTAL1 XTAL2) podłączono rezonator kwarcowy X1 o częstotliwości rezonansowej 25 MHz.

Jeżeli w aplikacji chcemy wykorzystywać zegar czasu rzeczywistego RTC, do wyprowadzeń RTXC1 oraz RTXC2 należy podłączyć rezonator X2 o częstotliwości 32,768 kHz.

Do linii VBAT podłączono baterię litową zapewniająca pracę zegara RTC oraz podtrzymanie zawartości wewnętrznej pamięci RAM podczas odłączenia zasilania głównego mikrokontrolera.

Jeżeli nie zależy nam na ciągłej pracy zegara RTC wyprowadzenie VBAT można dołączyć bezpośrednio do głównego napięcia zasilającego. Mikrokontrolery STR91x nie mają wbudowanego układu służącego do automatycznego zerowania po włączeniu napięcia zasilającego, dlatego do wejścia RESET należy podłączyć prosty układ RC lub specjalizowany układ zerujący. Pamięć Flash mikrokontrolerów STR91x w można programować za pomocą interfejsu JTAG, dlatego na płytce zastosowano również dwa złącza: ZL1 (Wiggler/OpenOCD) oraz ZL2 (Flash Link), służące do dołączenia interfejsu JTAG.

Lucjan Bryndza, EP
lucjan.bryndza@ep.com.pl